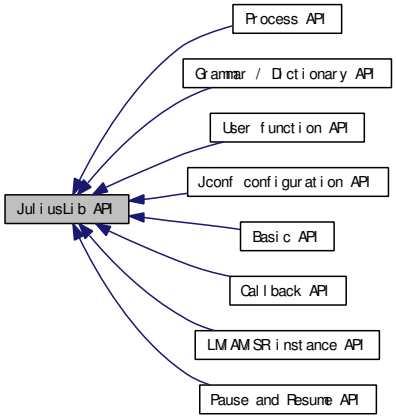
Chapter 6

Julius モジュール

6.1 JuliusLib API

JuliusLib API のコラボレーション図



Julius ライブラリ API 関数のリファレンスです.

モジュール

- Basic API
  認識エンジンの設定等
- Callback API
  認識結果やエンジン状態を知るためのコールバック
- Pause and Resume API
  エンジンの一時停止・再開

- User function API
  ユーザ関数の登録
- Process API
  モデルおよび認識プロセスの動的追加・削除・有効化・無効化
- Grammar / Dictionary API
  文法・単語辞書の操作
- Jconf configuration API
  Jconf **構造体によるパラメータ情報の管理**
- LM/AM/SR instance API
  モデルモジュールやプロセスを直接扱う関数.

6.1.1 説明

Julius ライブラリ API 関数のリファレンスです.

6.2 Basic API

Basic API のコラボレーション図

認識エンジンの設定等

関数

- void j_enable_debug_message ()
  JuliusLib 内の関数でデバッグメッセージをログに出力するようにする
- void j_disable_debug_message ()
  JuliusLib 内の関数でデバッグメッセージを出さないようにする.
- void j_enable_verbose_message ()
  JuliusLib 内の関数で主要メッセージをログに出力するようにする.
- void j_disable_verbose_message ()
  JuliusLib 内の関数で主要メッセージのログ出力をしないようにする.
- boolean j_adin_init (Recog *recog)
  設定で選択された *A/D-in* デバイスを初期化し認識の準備を行う.
- boolean j_adin_init_user (Recog *recog, void *arg)
  ユーザ定義の *A/D-in* 関数使用時の初期化関数.
- char * j_get_current_filename ()
  現在の入力ファイル名を返す.
- void j_recog_info (Recog *recog)
  エンジンの全設定と全システム情報をログに出力する.
- void j_output_argument_help (FILE *fp)
  ヘルプを表示する.
- int j_open_stream (Recog *recog, char *file_or_dev_name)
  音声入力ストリームを開く
- int j_recognize_stream (Recog *recog)
  入力ストリームの認識を行う
- boolean j_add_option (char *fmt, int argnum, int reqargnum, char *desc, boolean(*func)(Jconf *jconf, char *arg[], int argnum))
  Julius にユーザ定義オプションを追加する.

6.2.1 説明

認識エンジンの設定等

6.2.2 関数

6.2.2.1 boolean j_adin_init (Recog * *recog*)

設定で選択された A/D-in デバイスを初期化し認識の準備を行う.
そのデバイスに対して threading が指定されている場合は， A/D-in 用スレッドがここで開始される.

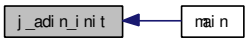
引数:

recog [in] engine instance

戻り値:

TRUE on success, FALSE on failure.

jfunc.c の 521 行で定義されています。
参照元 main().
呼出しグラフ:



6.2.2.2 boolean j_adin_init_user (Recog * *recog*, void * *arg*)

ユーザ定義の A/D-in 関数使用時の初期化関数.
デバイスの初期化を行わない 以外は j_adin_init() と同じである.

引数:

recog [in] engine instance
arg [in] argument

戻り値:

TRUE on success, else FALSE on error.

jfunc.c の 565 行で定義されています。

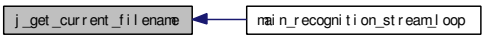
6.2.2.3 char* j_get_current_filename ()

現在の入力ファイル名を返す.
MFCC 入力時は使えない.

戻り値:

the file name.

jfunc.c の 602 行で定義されています。
参照元 main_recognition_stream_loop().
呼出しグラフ:



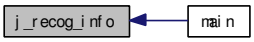
6.2.2.4 void j_recog_info (Recog * *recog*)

エンジンの全設定と全システム情報をログに出力する.

引数:

recog [in] engine instance

jfunc.c の 626 行で定義されています。
参照元 main().
呼出しグラフ:



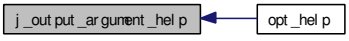
6.2.2.5 void j_output_argument_help (FILE * *fp*)

ヘルプを表示する.

引数:

fp [in] file pointer to output help

m_usage.c の 45 行で定義されています。
参照元 opt_help().
呼出しグラフ:



6.2.2.6 int j_open_stream (Recog * *recog*, char * *file_or_dev_name*)

音声入力ストリームを開く

引数:

recog [i/o] engine instance

file_or_dev_name [in] file or device name of the device

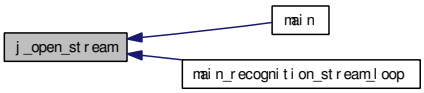
戻り値:

0 on success, -1 on error, -2 on decice initializatino error.

recogmain.c の 447 行で定義されています。

参照元 main(), と main_recognition_stream_loop().

呼出しグラフ:



6.2.2.7 int j_recognize_stream (Recog * *recog*)

入力ストリームの認識を行う

入力ストリームに対して （必要であれば）区間検出や VAD を行いながら認識を繰り返し行っていく. 入力が終端に達するかあるいはエラーで終了する.

アプリケーションから認識の中断をリクエストされたときは， CALLBACK_EVENT_PAUSE, CALLBACK_PAUSE_FUNCTION, CALLBACK_EVENT_RESUME の順に呼んだあと認識に戻る. このため， 認識を中断させている間に行う処理は，CALLBACK_PAUSE_FUNCTION に登録しておく必要がある. CALLBACK_PAUSE_FUNCTION に 登録されている全ての処理が終了したら認識を自動的に再開するので 注意すること.

引数:

recog [i/o] engine instance

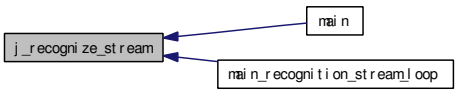
戻り値:

0 when finished recognizing all the input stream to the end, or -1 on error.

recogmain.c の 1216 行で定義されています。

参照元 main(), と main_recognition_stream_loop().

呼出しグラフ:



6.2.2.8 boolean j_add_option (char * *fmt*, int *argnum*, int *reqargnum*, char * *desc*, boolean(*)(Jconf *jconf, char *arg[], int argnum) *func*)

Julius にユーザ定義オプションを追加する.

argnum には引数の最大数，reqargnum はそのうち必須である引数の数を 指定する． argnum > reqargnum の場合，先頭から reqargnum 個が必須で， それ以降が optional として扱われる.

引数:

fmt [in] option string (should begin with '-')

argnum [in] total number of argument for this option (including optional)

reqargnum [in] number of required argument

desc [in] description string for help

func [in] option handling function

戻り値:

TRUE on success, FALSE on failure

useropt.c の 129 行で定義されています。

6.3 Callback API

Callback API のコラボレーション図

認識結果やエンジン状態を知るためのコールバック

関数

- int callback_add (Recog *recog, int code, void(*func)(Recog *recog, void *data), void *data)
  関数をコールバックレジストリに登録する.
- int callback_add_adin (Recog *recog, int code, void(*func)(Recog *recog, SP16 *buf, int len, void *data), void *data)
  関数を *A/D-in* **タイプのコールバックレジストリに登録する**.
- boolean callback_exist (Recog *recog, int code)
  コールバックレジストリに１つでも関数が登録されたかどうかを返す.
- boolean callback_delete (Recog *recog, int id)
  コールバックから関数を削除する.

6.3.1 説明

認識結果やエンジン状態を知るためのコールバック

6.3.2 関数

6.3.2.1 int callback_add (Recog * *recog*, int *code*, void(*)(Recog *recog, void *data) *func*, void * *data*)

関数をコールバックレジストリに登録する.

引数:

recog [i/o] engine instance
code [in] code in which the function will be registered
func [in] function
data [in] user-specified argument to be passed when the function is called inside Julius

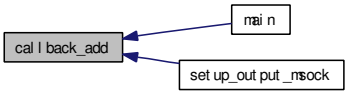
戻り値:

global callback ID unique for the whole process, or -1 on error.

callback.c の 135 行で定義されています。

参照元 main(), と setup_output_msock().

呼出しグラフ:



6.3.2.2 int callback_add_adin (Recog * *recog*, int *code*, void(*)(Recog *recog, SP16 *buf, int len, void *data) *func*, void * *data*)

関数を A/D-in タイプのコールバックレジストリに登録する.

引数:

recog [i/o] engine instance

code [in] code in which the function will be registered

func [in] function

data [in] user-specified argument to be passed when the function is called inside Julius

戻り値:

global callback ID unique for the whole process, or -1 on error.

callback.c の 161 行で定義されています。

6.3.2.3 boolean callback_exist (Recog * *recog*, int *code*)

コールバックレジストリに１つでも関数が登録されたかどうかを返す.

引数:

recog [in] engine instance

code [in] callback code

戻り値:

TRUE when at least one is registered, or FALSE if none.

callback.c の 246 行で定義されています。

6.3.2.4 boolean callback_delete (Recog * *recog*, int *id*)

コールバックから関数を削除する.

引数:

recog [i/o] engine instance
id [in] global callback ID to delete

戻り値:

TRUE on success, or FALSE on failure.

callback.c の 271 行で定義されています。

6.4 Pause and Resume API

Pause and Resume API のコラボレーション図

エンジンの一時停止・再開

関数

- void j_request_pause (Recog *recog)
  エンジンに認識処理を一時停止するよう要求する.
- void j_request_terminate (Recog *recog)
  エンジンに認識処理を即時停止するよう要求する.
- void j_request_resume (Recog *recog)
  一時停止しているエンジンを再開させる.

6.4.1 説明

エンジンの一時停止・再開

6.4.2 関数

6.4.2.1 void j_request_pause (Recog * *recog*)

エンジンに認識処理を一時停止するよう要求する.
この関数を呼出し時に 音声入力を実行中であった場合，その入力の認識が終了したあとで停止する.

引数:

recog [in] engine instance

jfunc.c の 52 行で定義されています。

6.4.2.2 void j_request_terminate (Recog * *recog*)

エンジンに認識処理を即時停止するよう要求する.
この関数を呼出し時に 音声入力を実行中の場合，その入力を破棄して即座に停止する.

引数:

recog [in] engine instance

jfunc.c の 85 行で定義されています。

6.4.2.3 void j_request_resume (Recog * *recog*)

一時停止しているエンジンを再開させる.

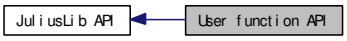
引数:

recog

jfunc.c の 117 行で定義されています。

6.5 User function API

User function API のコラボレーション図



ユーザ関数の登録

関数

- boolean j_regist_user_lm_func (PROCESS_LM *lm, LOGPROB(*unifunc)(WORD_INFO *winfo, WORD_ID w, LOGPROB ngram_prob), LOGPROB(*bifunc)(WORD_INFO *winfo, WORD_ID context, WORD_ID w, LOGPROB ngram_prob), LOGPROB(*probfunc)(WORD_INFO *winfo, WORD_ID *contexts, int context_len, WORD_ID w, LOGPROB ngram_prob))

  言語モデル処理インスタンスにユーザ定義の言語スコア付与関数を登録する.

- boolean j_regist_user_param_func (Recog *recog, boolean(*user_calc_vector)(MFCCCalc *, SP16 *, int))

  ユーザ定義の特徴量計算関数を使うようエンジンに登録する.

6.5.1 説明

ユーザ関数の登録

6.5.2 関数

6.5.2.1 boolean j_regist_user_lm_func (PROCESS_LM * *lm*, LOGPROB(*)(WORD_INFO *winfo, WORD_ID w, LOGPROB ngram_prob) *unifunc*, LOGPROB(*)(WORD_INFO *winfo, WORD_ID context, WORD_ID w, LOGPROB ngram_prob) *bifunc*, LOGPROB(*)(WORD_INFO *winfo, WORD_ID *contexts, int context_len, WORD_ID w, LOGPROB ngram_prob) *probfunc*)

言語モデル処理インスタンスにユーザ定義の言語スコア付与関数を登録する.

この関数はエンジンインスタンス生成後から j_final_fusion() が呼ばれる までの間に呼ぶ必要がある. 注意：ユーザ定義の言語スコア関数を使う場合は 実行時オプション "-userlm" も指定する必要があることに注意せよ.

引数:

lm [i/o] LM processing instance

unifunc [in] pointer to the user-defined unigram function
bifunc [in] pointer to the user-defined bi-igram function
probfunc [in] pointer to the user-defined N-gram function

戻り値:

TRUE on success, FALSE on failure.

jfunc.c の 719 行で定義されています。

6.5.2.2 boolean j_regist_user_param_func (Recog * *recog*, boolean(*)(MFCCCalc *, SP16 *, int) *user_calc_vector*)

ユーザ定義の特徴量計算関数を使うようエンジンに登録する.

引数:

recog [i/o] engine instance
user_calc_vector [in] pointer to function of parameter extraction

戻り値:

TRUE on success, FALSE on error.

jfunc.c の 748 行で定義されています。

6.6 Process API

Process API のコラボレーション図

モデルおよび認識プロセスの動的追加・削除・有効化・無効化

関数

- boolean j_process_deactivate (Recog *recog, char *name)
  指定された名前の認識処理インスタンスの動作を一時停止させる.
- boolean j_process_deactivate_by_id (Recog *recog, int id)
  指定された認識処理インスタンスの動作を一時停止させる.
- boolean j_process_activate (Recog *recog, char *name)
  一時停止されていた認識処理インスタンスの動作を再開させる.
- boolean j_process_activate_by_id (Recog *recog, int id)
  一時停止されていた認識処理インスタンスの動作を再開させる *(ID* **指定***)*.
- boolean j_process_add_lm (Recog *recog, JCONF_LM *lmconf, JCONF_SEARCH *sconf, char *name)
  LM **および** *SR* **設定に基づき認識処理プロセスを追加する**.
- boolean j_process_remove (Recog *recog, JCONF_SEARCH *sconf)
  認識処理インスタンスを削除する.
- boolean j_process_lm_remove (Recog *recog, JCONF_LM *lmconf)
  言語モデルインスタンスを削除する.
- boolean j_process_am_remove (Recog *recog, JCONF_AM *amconf)
  言語モデルインスタンスを削除する（実験中）.

6.6.1 説明

モデルおよび認識プロセスの動的追加・削除・有効化・無効化

6.6.2 関数

6.6.2.1 boolean j_process_deactivate (Recog * *recog*, char * *name*)

指定された名前の認識処理インスタンスの動作を一時停止させる.

実際に停止するのは次の音声認識の合間である.

引数:

recog [i/o] engine instance

name [in] SR name to deactivate

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1010 行で定義されています。

6.6.2.2 boolean j_process_deactivate_by_id (Recog * *recog*, int *id*)

指定された認識処理インスタンスの動作を一時停止させる.

対象インスタンスを ID 番号で指定する場合はこちらを使う. 実際に停止するのは次の音声認識の合間である.

引数:

recog [i/o] engine instance

id [in] SR ID to deactivate

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1054 行で定義されています。

6.6.2.3 boolean j_process_activate (Recog * *recog*, char * *name*)

一時停止されていた認識処理インスタンスの動作を再開させる.

実際に再開するのは次の音声認識の合間である.

引数:

recog [i/o] engine instance

name [in] SR name to activate

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1098 行で定義されています。

6.6.2.4 boolean j_process_activate_by_id (Recog * *recog*, int *id*)

一時停止されていた認識処理インスタンスの動作を再開させる (ID 指定).

実際に再開するのは次の音声認識の合間である.

引数:

recog [i/o] engine instance
id [in] SR ID to activate

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1142 行で定義されています。

6.6.2.5 boolean j_process_add_lm (Recog * *recog*, JCONF_LM * *lmconf*, JCONF_SEARCH * *sconf*, char * *name*)

LM および SR 設定に基づき認識処理プロセスを追加する.

この関数は与えられた LM 設定および SR 設定データに基づき，新たな LM インスタンスおよび認識プロセスインスタンスをエンジン内部に 生成する. AM については現在のデフォルト AM が自動的に用いられる. 名前は LM インスタンス，認識プロセスインスタンスとも同じ名前が あたえられる.

引数:

recog [i/o] engine instance
lmconf [in] a new LM configuration
sconf [in] a new SR configuration
name [in] name of the new instances

戻り値:

TRUE on success, FALSE on error.

jfunc.c の 1195 行で定義されています。

6.6.2.6 boolean j_process_remove (Recog * *recog*, JCONF_SEARCH * *sconf*)

認識処理インスタンスを削除する.

指定された SEARCH 設定もこの関数内で解放・削除される.

引数:

recog [in] engine instance
sconf [in] SEARCH configuration corresponding to the target recognition process to remove

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1258 行で定義されています。

6.6.2.7 boolean j_process_lm_remove (Recog * *recog*, JCONF_LM * *lmconf*)

言語モデルインスタンスを削除する.
指定された言語モデル設定もこの関数内で解放・削除される.

引数:

recog [in] engine instance
lmconf [in] LM configuration corresponding to the target LM process to remove

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1336 行で定義されています。

6.6.2.8 boolean j_process_am_remove (Recog * *recog*, JCONF_AM * *amconf*)

言語モデルインスタンスを削除する（実験中）.
指定された言語モデル設定もこの関数内で解放・削除される.

引数:

recog [in] engine instance
amconf [in] AM configuration corresponding to the target AM process to remove

戻り値:

TRUE on success, or FALSE on failure.

jfunc.c の 1424 行で定義されています。

6.7 Grammar / Dictionary API

Grammar / Dictionary API のコラボレーション図

文法・単語辞書の操作

関数

- void multigram_add_gramlist (char *dfafile, char *dictfile, JCONF_LM *j, int lmvar)
  起動時読み込みリストに文法を追加する.
- void multigram_remove_gramlist (JCONF_LM *j)
  起動時読み込みリストを消す.
- boolean multigram_add_prefix_list (char *prefix_list, char *cwd, JCONF_LM *j, int lmvar)
  プレフィックスから複数の文法を起動時読み込みリストに追加する.
- boolean multigram_add_prefix_filelist (char *listfile, JCONF_LM *j, int lmvar)
  リストファイルを読み込み複数文法を起動時読み込みリストに追加する.
- void schedule_grammar_update (Recog *recog)
  全文法の変更をチェックし，必要であれば認識用辞書を再構築するよう エンジンに要求する.
- void j_reset_reload (Recog *recog)
  再構築要求フラグをクリアする.
- boolean multigram_build (RecogProcess *r)
  グローバル文法を調べ，必要があれば木構造化辞書を（再）構築する.
- void multigram_add (DFA_INFO *dfa, WORD_INFO *winfo, char *name, PROCESS_LM *lm)
  新たな文法を，文法リストに追加する.
- boolean multigram_delete (int delid, PROCESS_LM *lm)
  文法を削除する.
- void multigram_delete_all (PROCESS_LM *lm)
  すべての文法を次回更新時に削除するようマークする.
- int multigram_activate (int gid, PROCESS_LM *lm)
  文法を有効化する.

- int multigram_deactivate (int gid, PROCESS_LM *lm)
  文法を無効化する.
- boolean multigram_update (PROCESS_LM *lm)
  グローバル文法の更新
- int multigram_get_all_num (PROCESS_LM *lm)
  現在ある文法の数を得る *(active/inactive* **とも***)*.
- int multigram_get_gram_from_category (int category, PROCESS_LM *lm)
  単語カテゴリの属する文法を得る.
- MULTIGRAM * multigram_get_grammar_by_name (PROCESS_LM *lm, char *gramname)
  LM **中の文法を名前で検索する**.
- MULTIGRAM * multigram_get_grammar_by_id (PROCESS_LM *lm, unsigned short id)
  LM **中の文法を** *ID* **番号で検索する**.
- boolean multigram_add_words_to_grammar (PROCESS_LM *lm, MULTIGRAM *m, WORD_INFO *winfo)
  単語集合を文法に追加する．
- boolean multigram_add_words_to_grammar_by_name (PROCESS_LM *lm, char *gramname, WORD_INFO *winfo)
  名前で指定された文法に単語集合を追加する．
- boolean multigram_add_words_to_grammar_by_id (PROCESS_LM *lm, unsigned short id, WORD_INFO *winfo)
  番号で指定された文法に単語集合を追加する．

6.7.1 説明

文法・単語辞書の操作

6.7.2 関数

6.7.2.1 void multigram_add_gramlist (char * *dfafile*, char * *dictfile*, JCONF_LM * *j*, int *lmvar*)

起動時読み込みリストに文法を追加する.

引数:

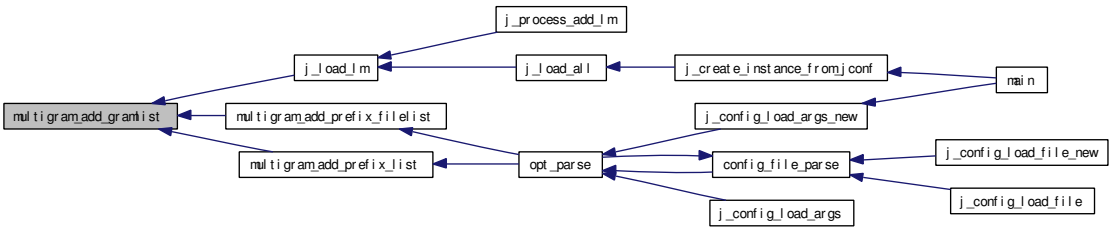
dfafile [in] DFA ファイル

dictfile [in] 単語辞書
j [in] LM 設定パラメータ
lmvar [in] LM 詳細型 id

gramlist.c の 66 行で定義されています。

参照元 j_load_lm(), multigram_add_prefix_filelist(), と multigram_add_prefix_list().

呼出しグラフ:



6.7.2.2 void multigram_remove_gramlist (JCONF_LM * *j*)

起動時読み込みリストを消す.

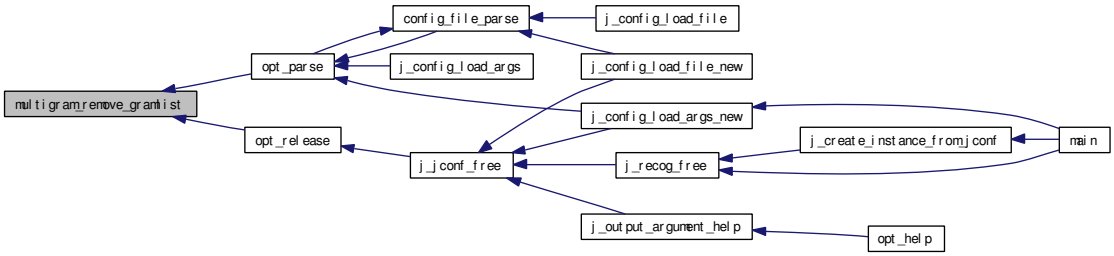
引数:

j [in] LM 設定パラメータ

gramlist.c の 103 行で定義されています。

参照元 opt_parse(), と opt_release().

呼出しグラフ:



6.7.2.3 boolean multigram_add_prefix_list (char * *prefix_list*, char * *cwd*, JCONF_LM * *j*, int *lmvar*)

プレフィックスから複数の文法を起動時読み込みリストに追加する.

プレフィックスは "foo", あるいは "foo,bar" のようにコンマ区切りで 複数与えることができます. 各文字列の後ろに ".dfa", ".dict" をつけた ファイルを，それぞれ文法ファイル・辞書ファイルとして順次読み込みます. 読み込まれた文法は順次，文法リストに追加されます.

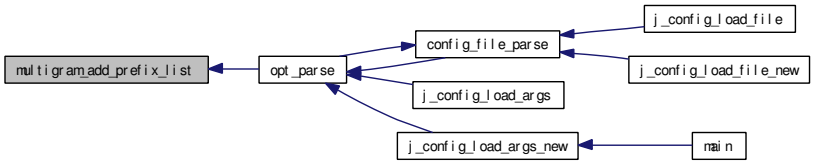
引数:

prefix_list [in] プレフィックスのリスト
cwd [in] カレントディレクトリの文字列
j [in] LM 設定パラメータ
lmvar [in] LM 詳細型 id

gramlist.c の 163 行で定義されています。

参照元 opt_parse().

呼出しグラフ:



6.7.2.4 boolean multigram_add_prefix_filelist (char * *listfile*, JCONF_LM * *j*, int *lmvar*)

リストファイルを読み込み複数文法を起動時読み込みリストに追加する.

ファイル内に 1 行に 1 つずつ記述された文法のプレフィックスから, 対応する文法ファイルを順次読み込みます.

各行の文字列の後ろに ".dfa", ".dict" をつけたファイルを， それぞれ文法ファイル・辞書ファイルとして順次読み込みます. 読み込まれた文法は順次，文法リストに追加されます.

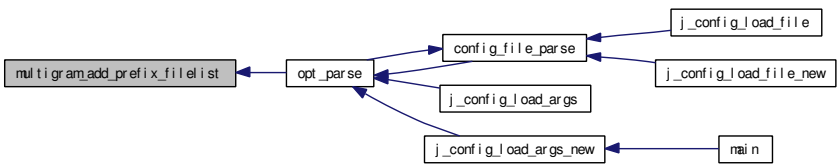
引数:

listfile [in] プレフィックスリストのファイル名
j [in] LM 設定パラメータ
lmvar [in] LM 詳細型 id

gramlist.c の 265 行で定義されています。

参照元 opt_parse().

呼出しグラフ:



6.7.2.5 void schedule_grammar_update (Recog * *recog*)

全文法の変更をチェックし，必要であれば認識用辞書を再構築するよう エンジンに要求する.

実際の処理は次の認識の合間に行われる. この関数は文法を追加したり削除したなど， 文法リストに変更を加えたあとに必ず呼ぶべきである.

引数:

recog [in] engine instance

jfunc.c の 152 行で定義されています。

6.7.2.6 void j_reset_reload (Recog * *recog*)

再構築要求フラグをクリアする.

引数:

recog [in] engine instance

jfunc.c の 197 行で定義されています。

6.7.2.7 boolean multigram_build (RecogProcess * *r*)

グローバル文法を調べ，必要があれば木構造化辞書を（再）構築する.

グローバル辞書に変更があれば，その更新されたグローバル 辞書から木構造化辞書などの音声認識用データ構造を再構築する.

引数:

r [in] recognition process instance

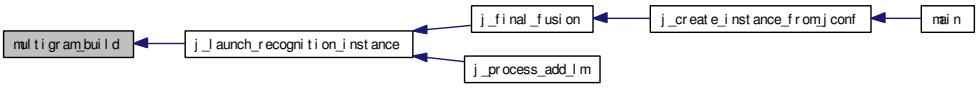
戻り値:

TRUE on success, FALSE on error.

multi-gram.c の 173 行で定義されています。

参照元 j_launch_recognition_instance().

呼出しグラフ:



6.7.2.8 void multigram_add (DFA_INFO * *dfa*, WORD_INFO * *winfo*, char * *name*, PROCESS_LM * *lm*)

新たな文法を，文法リストに追加する.

現在インスタンスが保持している文法のリストは lm->grammars に保存される．追加した文法には，newbie, active のフラグがセットされ，次回の 文法更新チェック時に更新対象となる．

引数:

dfa [in] 追加登録する文法の DFA 情報

winfo [in] 追加登録する文法の辞書情報

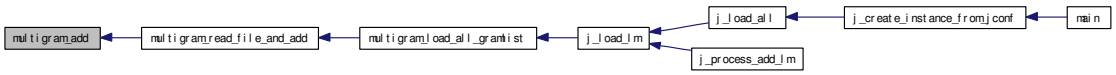
name [in] 追加登録する文法の名称

lm [i/o] 言語処理インスタンス

multi-gram.c の 273 行で定義されています。

参照元 multigram_read_file_and_add().

呼出しグラフ:



6.7.2.9 boolean multigram_delete (int *delid*, PROCESS_LM * *lm*)

文法を削除する．

文法リスト中のある文法について，削除マークを付ける．実際の削除は multigram_exec_delete() で行われる．

引数:

delid [in] 削除する文法の文法 ID

lm [i/o] 言語処理インスタンス

戻り値:

通常時 TRUE を返す．指定された ID の文法が無い場合は FALSE を返す．

multi-gram.c の 326 行で定義されています。

6.7.2.10 void multigram_delete_all (PROCESS_LM * *lm*)

すべての文法を次回更新時に削除するようマークする．

引数:

lm [i/o] 言語処理インスタンス

multi-gram.c の 359 行で定義されています。

6.7.2.11 int multigram_activate (int *gid*, PROCESS_LM * *lm*)

文法を有効化する.

ここでは次回更新時に 反映されるようにマークをつけるのみである.

引数:

gid [in] 有効化したい文法の ID
lm [i/o] 言語処理インスタンス

multi-gram.c の 441 行で定義されています。

6.7.2.12 int multigram_deactivate (int *gid*, PROCESS_LM * *lm*)

文法を無効化する.

無効化された文法は 認識において仮説展開されない. これによって，グローバル辞書を 再構築することなく，一時的に個々の文法を ON/OFF できる. 無効化した 文法は multigram_activate() で再び有効化できる. なおここでは 次回の文法更新タイミングで反映されるようにマークをつけるのみである.

引数:

gid [in] 無効化したい文法の ID
lm [i/o] 言語処理インスタンス

multi-gram.c の 503 行で定義されています。

6.7.2.13 boolean multigram_update (PROCESS_LM * *lm*)

グローバル文法の更新

前回呼出しからの文法リストの変更をチェックする. リスト中に削除マークがつけられた文法がある場合は，その文法を削除し， グローバル辞書を再構築する. 新たに追加された文法がある場合は，その文法を現在のグローバル辞書の末尾に追加する.

引数:

lm [i/o] 言語処理インスタンス

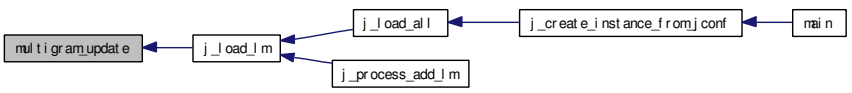
戻り値:

常に TRUE を返す.

multi-gram.c の 615 行で定義されています。

参照元 j_load_lm().

呼出しグラフ:



6.7.2.14 int multigram_get_all_num (PROCESS_LM * *lm*)

現在ある文法の数を得る (active/inactive とも).

引数:

lm [i/o] 言語処理インスタンス

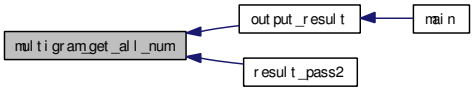
戻り値:

文法の数を返す.

multi-gram.c の 923 行で定義されています。

参照元 output_result(), と result_pass2().

呼出しグラフ:



6.7.2.15 int multigram_get_gram_from_category (int *category*, PROCESS_LM * *lm*)

単語カテゴリの属する文法を得る.

引数:

category 単語カテゴリ ID
lm [i/o] 言語処理インスタンス

戻り値:

単語カテゴリの属する文法の ID を返す.

multi-gram.c の 955 行で定義されています。

6.7.2.16 MULTIGRAM* multigram_get_grammar_by_name (PROCESS_LM * *lm*, char * *gramname*)

LM 中の文法を名前で検索する.

引数:

lm [in] LM process instance
gramname [in] grammar name

戻り値:

poitner to the grammar, or NULL if not found.

multi-gram.c の 1019 行で定義されています。

参照元 multigram_add_words_to_grammar_by_name().

呼出しグラフ:

6.7.2.17 MULTIGRAM* multigram_get_grammar_by_id (PROCESS_LM * *lm*, unsigned short *id*)

LM 中の文法を ID 番号で検索する.

引数:

lm [in] LM process instance

id [in] ID number

戻り値:

poitner to the grammar, or NULL if not found.

multi-gram.c の 1052 行で定義されています。

参照元 multigram_add_words_to_grammar_by_id().

呼出しグラフ:

6.7.2.18 boolean multigram_add_words_to_grammar (PROCESS_LM * *lm*, MULTIGRAM * *m*, WORD_INFO * *winfo*)

単語集合を文法に追加する.

追加する単語の文法カテゴリ ID については，すでにアサインされているものが そのままコピーされる．よって，それらはこの関数を呼び出す前に， 追加対象の文法で整合性が取れるよう正しく設定されている必要がある． 木構造化辞書全体が，後に再構築される.

単語 N-gram 言語モデルへの辞書追加は現在サポートされていない.

引数:

lm [i/o] LM process instance

m [i/o] grammar to which the winfo will be appended

winfo [in] words to be added to the grammar

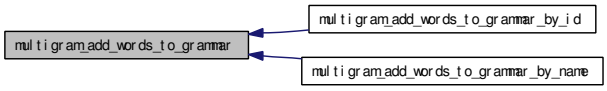
戻り値:

TRUE on success, or FALSE on failure.

multi-gram.c の 1101 行で定義されています。

参照元 multigram_add_words_to_grammar_by_id(), と multigram_add_words_to_grammar_by_name().

呼出しグラフ:



6.7.2.19 boolean multigram_add_words_to_grammar_by_name (PROCESS_LM * *lm*, char * *gramname*, WORD_INFO * *winfo*)

名前で指定された文法に単語集合を追加する.

multigram_add_words_to_grammar() を文法名で指定して実行する.

引数:

lm [i/o] LM process instance
gramname [in] name of the grammar to which the winfo will be appended
winfo [in] words to be added to the grammar

戻り値:

TRUE on success, or FALSE on failure.

multi-gram.c の 1154 行で定義されています。

6.7.2.20 boolean multigram_add_words_to_grammar_by_id (PROCESS_LM * *lm*, unsigned short *id*, WORD_INFO * *winfo*)

番号で指定された文法に単語集合を追加する.

multigram_add_words_to_grammar() を番号で指定して実行する.

引数:

lm [i/o] LM process instance
id [in] ID number of the grammar to which the winfo will be appended
winfo [in] words to be added to the grammar

戻り値:

TRUE on success, or FALSE on failure.

multi-gram.c の 1185 行で定義されています。

6.8 Jconf configuration API

Jconf configuration API のコラボレーション図

Jconf 構造体によるパラメータ情報の管理

関数

- JCONF_AM * j_jconf_am_new ()
  音響モデル *(AM)* **パラメータ構造体を新たに割り付ける**.
- void j_jconf_am_free (JCONF_AM *amconf)
  音響モデル *(AM)* **パラメータ構造体を解放する**.
- boolean j_jconf_am_regist (Jconf *jconf, JCONF_AM *amconf, char *name)
  音響モデル *(AM)* **パラメータ構造体を** *jconf* **に登録する．** *jconf* **内に同じ名前のモジュールが既に登録されている場合はエラーとなる．**
- JCONF_LM * j_jconf_lm_new ()
  言語モデル *(LM)* **パラメータ構造体を新たに割り付ける 内部メンバにはデフォルト値が格納される**.
- void j_jconf_lm_free (JCONF_LM *lmconf)
  言語モデル *(LM)* **パラメータ構造体を解放する**
- boolean j_jconf_lm_regist (Jconf *jconf, JCONF_LM *lmconf, char *name)
  言語モデル *(LM)* **パラメータ構造体を** *jconf* **に登録する．** *jconf* **内に同じ名前のモジュールが既に登録されている場合はエラーとなる．**
- JCONF_SEARCH * j_jconf_search_new ()
  探索パラメータ *(SEARCH)* **構造体を新たに割り付ける**.
- void j_jconf_search_free (JCONF_SEARCH *sconf)
  探索パラメータ *(SEARCH)* **構造体を解放する**
- boolean j_jconf_search_regist (Jconf *jconf, JCONF_SEARCH *sconf, char *name)
  探索 *(SEARCH)* **パラメータ構造体を** *jconf* **に登録する．** *jconf* **内に同じ名前のモジュールが既に登録されている場合はエラーとなる．**
- Jconf * j_jconf_new ()
  全体のパラメータ構造体を新たに割り付ける.
- void j_jconf_free (Jconf *jconf)
  全体のパラメータ構造体を開放する.

- int j_config_load_args (Jconf *jconf, int argc, char *argv[])
  コマンド引数からパラメータを読み込み，*jconf* 内の各設定インスタンスに 値を格納する.
- int j_config_load_file (Jconf *jconf, char *filename)
  ***jconf* ファイルからパラメータを読み込み，*jconf* 内の各設定インスタンスに 値を格納する**.
- Jconf * j_config_load_args_new (int argc, char *argv[])
  コマンド引数からパラメータを読み込み，その値を格納した 新たな設定インスタンスを割り付けて返す.
- Jconf * j_config_load_file_new (char *filename)
  新たな設定インスタンスを割り付け，そこに *jconf* ファイルから設定パラメータを読み込んで返す.
- JCONF_AM * j_get_amconf_by_name (Jconf *jconf, char *name)
  ***jconf* 内の *AM* モジュール設定構造体を名前で検索する**.
- JCONF_AM * j_get_amconf_by_id (Jconf *jconf, int id)
  ***jconf* 内の *AM* モジュール設定構造体を *ID* で検索する**.
- JCONF_AM * j_get_amconf_default (Jconf *jconf)
  デフォルトの *AM* 設定を返す.
- JCONF_LM * j_get_lmconf_by_name (Jconf *jconf, char *name)
  ***jconf* 内の *LM* モジュール設定構造体を名前で検索する**.
- JCONF_LM * j_get_lmconf_by_id (Jconf *jconf, int id)
  ***jconf* 内の *LM* モジュール設定構造体を *ID* で検索する**.
- JCONF_SEARCH * j_get_searchconf_by_name (Jconf *jconf, char *name)
  ***jconf* 内の *SESARCH* モジュール設定構造体を名前で検索する**.
- JCONF_SEARCH * j_get_searchconf_by_id (Jconf *jconf, int id)
  ***jconf* 内の *SEARCH* モジュール設定構造体を *ID* で検索する**.
- boolean j_jconf_finalize (Jconf *jconf)
  ***jconf* 設定パラメータを最終的に決定する**

6.8.1 説明

Jconf 構造体によるパラメータ情報の管理

6.8.2 関数

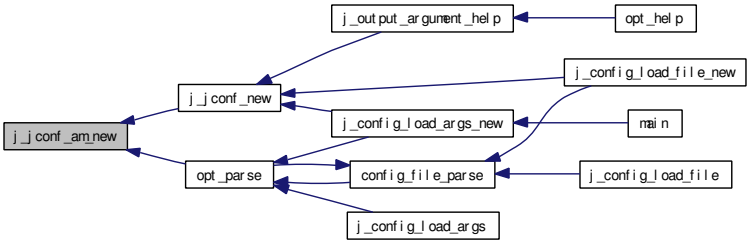
6.8.2.1 JCONF_AM* j_jconf_am_new ()

音響モデル (AM) パラメータ構造体を新たに割り付ける.
内部メンバにはデフォルト値が格納される.

戻り値:

the newly allocated AM parameter structure

instance.c の 336 行で定義されています。
参照元 j_jconf_new(), と opt_parse().
呼出しグラフ:



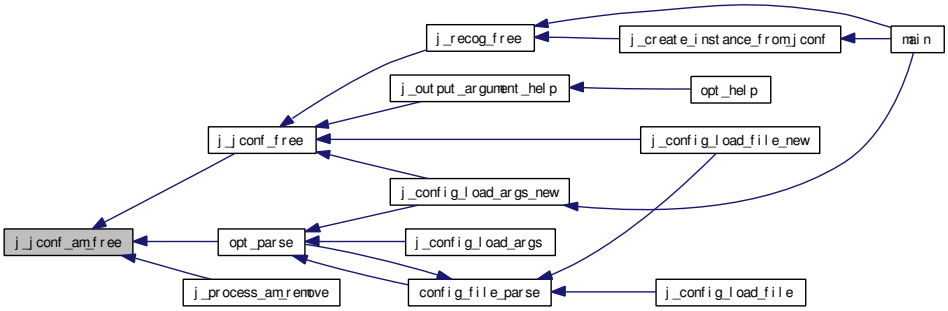
6.8.2.2 void j_jconf_am_free (JCONF_AM * *amconf*)

音響モデル (AM) パラメータ構造体を解放する.
内部メンバにはデフォルト値が格納される.

引数:

amconf [in] AM configuration

instance.c の 363 行で定義されています。
参照元 j_jconf_free(), j_process_am_remove(), と opt_parse().
呼出しグラフ:



6.8.2.3 boolean j_jconf_am_regist (Jconf * *jconf*, JCONF_AM * *amconf*, char * *name*)

音響モデル (AM) パラメータ構造体を jconf に登録する． jconf 内に同じ名前のモジュールが既に登録されている場合はエラーとなる．

引数:

jconf [i/o] global jconf
amconf [in] AM configuration to register
name [in] module name

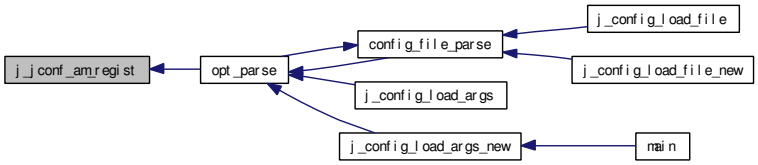
戻り値:

TRUE on success, FALSE on failure

instance.c の 389 行で定義されています。

参照元 opt_parse().

呼出しグラフ:



6.8.2.4 JCONF_LM* j_jconf_lm_new ()

言語モデル (LM) パラメータ構造体を新たに割り付ける 内部メンバにはデフォルト値が格納される．

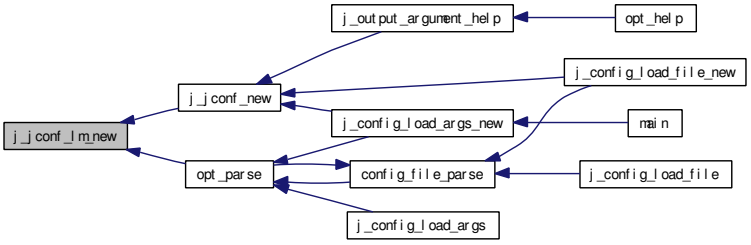
戻り値:

the newly allocated LM parameter structure.

instance.c の 440 行で定義されています。

参照元 j_jconf_new(), と opt_parse().

呼出しグラフ:



6.8.2.5 void j_jconf_lm_free (JCONF_LM * *lmconf*)

言語モデル (LM) パラメータ構造体を解放する

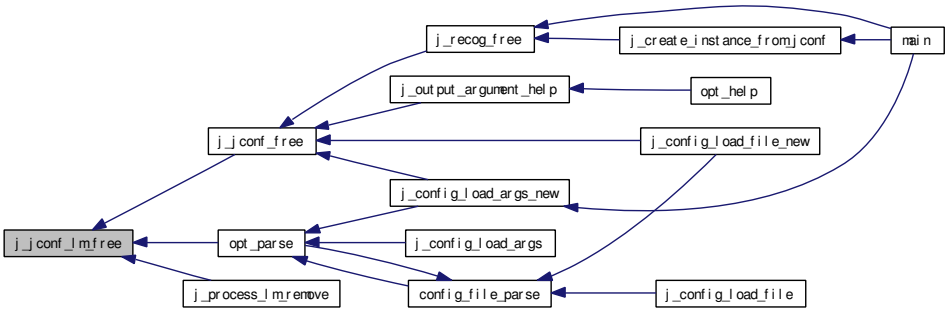
引数:

lmconf [in] LM parameter structure

instance.c の 465 行で定義されています。

参照元 j_jconf_free(), j_process_lm_remove(), と opt_parse().

呼出しグラフ:



6.8.2.6 boolean j_jconf_lm_regist (Jconf * *jconf*, JCONF_LM * *lmconf*, char * *name*)

言語モデル (LM) パラメータ構造体を jconf に登録する． jconf 内に同じ名前のモジュールが既に登録されている場合はエラーとなる．

引数:

jconf [i/o] global jconf
lmconf [in] LM configuration to register
name [in] module name

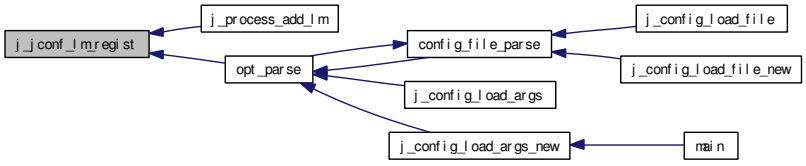
戻り値:

TRUE on success, FALSE on failure

instance.c の 491 行で定義されています。

参照元 j_process_add_lm(), と opt_parse().

呼出しグラフ:



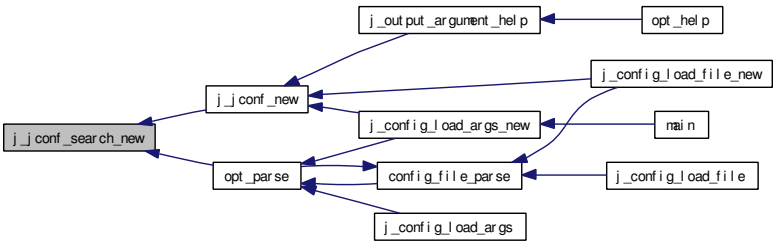
6.8.2.7 JCONF_SEARCH* j_jconf_search_new ()

探索パラメータ (SEARCH) 構造体を新たに割り付ける.
内部メンバにはデフォルト値が格納される.

戻り値:

the newly allocated SEARCH parameter structure.

instance.c の 542 行で定義されています。
参照元 j_jconf_new(), と opt_parse().
呼出しグラフ:



6.8.2.8 void j_jconf_search_free (JCONF_SEARCH * *sconf*)

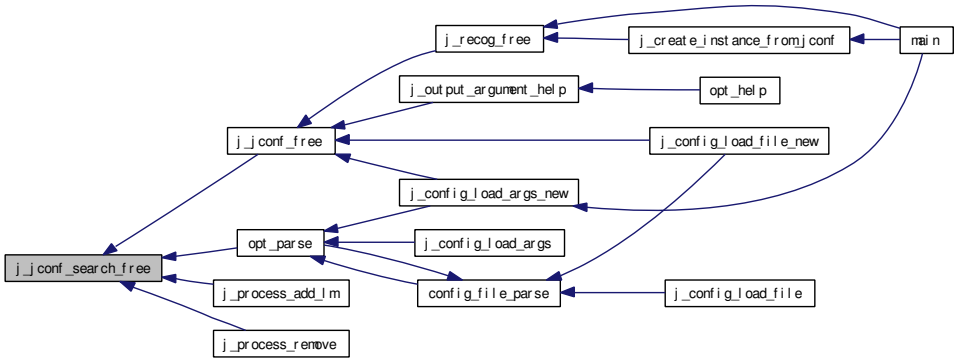
探索パラメータ (SEARCH) 構造体を解放する

引数:

sconf [in] SEARCH parameter structure

instance.c の 567 行で定義されています。
参照元 j_jconf_free(), j_process_add_lm(), j_process_remove(), と opt_parse().

呼出しグラフ:



6.8.2.9 boolean j_jconf_search_regist (Jconf * *jconf*, JCONF_SEARCH * *sconf*, char * *name*)

探索 (SEARCH) パラメータ構造体を jconf に登録する． jconf 内に同じ名前のモジュールが既に登録されている場合はエラーとなる．

引数:

jconf [i/o] global jconf
sconf [in] SEARCH configuration to register
name [in] module name

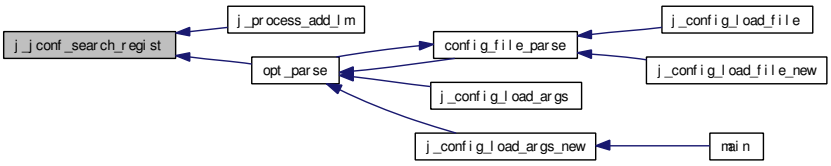
戻り値:

TRUE on success, FALSE on failure

instance.c の 593 行で定義されています。

参照元 j_process_add_lm(), と opt_parse().

呼出しグラフ:



6.8.2.10 Jconf* j_jconf_new ()

全体のパラメータ構造体を新たに割り付ける．

JCONF_AM, JCONF_LM, JCONF_SEARCH も 1 つづつ割り当てられる．これらは -AM 等の指定を含まない 3.x 以前の jconf を読み込んだときに， そのまま用いられる．

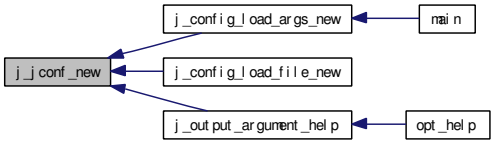
戻り値:

the newly allocated global configuration parameter structure.

instance.c の 649 行で定義されています。

参照元 j_config_load_args_new(), j_config_load_file_new(), と j_output_argument_help().

呼出しグラフ:



6.8.2.11 void j_jconf_free (Jconf * *jconf*)

全体のパラメータ構造体を開放する．

JCONF_AM, JCONF_LM, JCONF_SEARCH もすべて開放される．

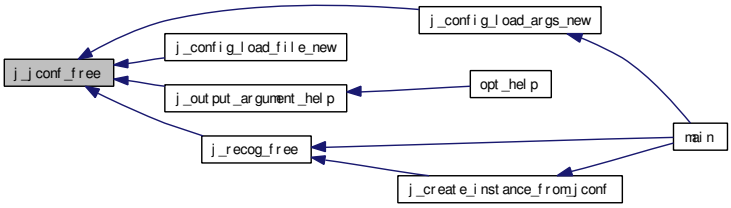
引数:

jconf [in] global configuration parameter structure

instance.c の 702 行で定義されています。

参照元 j_config_load_args_new(), j_config_load_file_new(), j_output_argument_help(), と j_recog_free().

呼出しグラフ:



6.8.2.12 int j_config_load_args (Jconf * *jconf*, int *argc*, char * *argv*[])

コマンド引数からパラメータを読み込み，jconf 内の各設定インスタンスに 値を格納する．

引数:

jconf [i/o] global configuration instance
argc [in] number of arguments
argv [in] list of argument strings

戻り値:

0 on success, or -1 on failure.

jfunc.c の 319 行で定義されています。

6.8.2.13 int j_config_load_file (Jconf * *jconf*, char * *filename*)

jconf ファイルからパラメータを読み込み，jconf 内の各設定インスタンスに 値を格納する.

引数:

jconf [i/o] glbal configuration instance
filename [in] jconf filename

戻り値:

0 on sucess, or -1 on failure.

jfunc.c の 368 行で定義されています。

6.8.2.14 Jconf* j_config_load_args_new (int *argc*, char * *argv*[])

コマンド引数からパラメータを読み込み，その値を格納した 新たな設定インスタンスを割り付けて返す.

引数:

argc [in] number of arguments
argv [in] list of argument strings

戻り値:

the newly allocated global configuration instance.

jfunc.c の 417 行で定義されています。

参照元 main().

呼出しグラフ:

6.8.2.15 Jconf* j_config_load_file_new (char * *filename*)

新たな設定インスタンスを割り付け，そこに jconf ファイルから設定パラメータを読み込んで返す.

引数:

filename [in] jconf filename

戻り値:

the newly allocated global configuration instance.

jfunc.c の 468 行で定義されています。

6.8.2.16 JCONF_AM* j_get_amconf_by_name (Jconf * *jconf*, char * *name*)

jconf 内の AM モジュール設定構造体を名前で検索する.

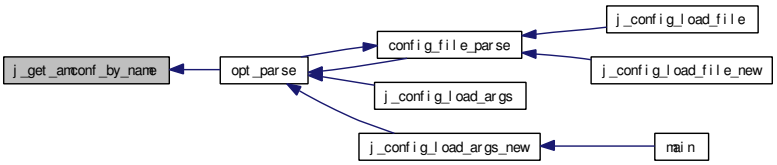
引数:

jconf [in] global configuration
name [in] AM module name

戻り値:

the specified AM configuration, or NULL if not found.

jfunc.c の 773 行で定義されています。
参照元 opt_parse().
呼出しグラフ:



6.8.2.17 JCONF_AM* j_get_amconf_by_id (Jconf * *jconf*, int *id*)

jconf 内の AM モジュール設定構造体を ID で検索する.

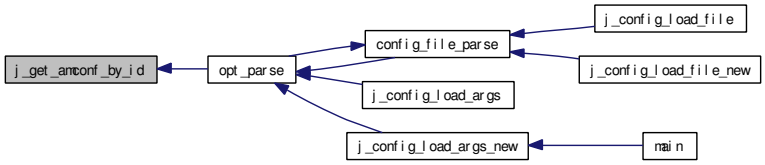
引数:

jconf [in] global configuration
id [in] AM module ID

戻り値:

the specified AM configuration, or NULL if not found.

jfunc.c の 807 行で定義されています。
参照元 opt_parse().
呼出しグラフ:



6.8.2.18 JCONF_AM* j_get_amconf_default (Jconf * *jconf*)

デフォルトの AM 設定を返す.
AM が複数設定されている場合，最も最近のものを返す.

引数:

jconf [in] global configuration

戻り値:

the specified AM configuration, or NULL if not found.

jfunc.c の 844 行で定義されています。
参照元 j_process_add_lm().
呼出しグラフ:

6.8.2.19 JCONF_LM* j_get_lmconf_by_name (Jconf * *jconf*, char * *name*)

jconf 内の LM モジュール設定構造体を名前で検索する.

引数:

jconf [in] global configuration
name [in] LM module name

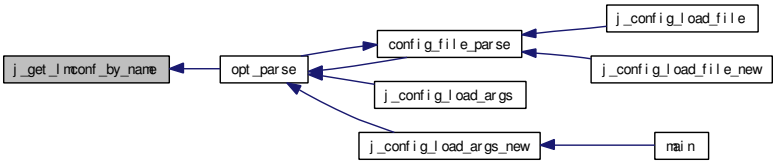
戻り値:

the specified LM configuration, or NULL if not found.

jfunc.c の 871 行で定義されています。

参照元 opt_parse().

呼出しグラフ:



6.8.2.20 JCONF_LM* j_get_lmconf_by_id (Jconf * *jconf*, int *id*)

jconf 内の LM モジュール設定構造体を ID で検索する.

引数:

jconf [in] global configuration

id [in] LM module ID

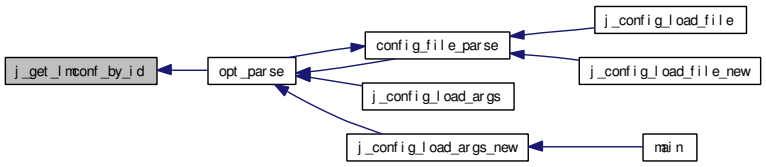
戻り値:

the specified LM configuration, or NULL if not found.

jfunc.c の 905 行で定義されています。

参照元 opt_parse().

呼出しグラフ:



6.8.2.21 JCONF_SEARCH* j_get_searchconf_by_name (Jconf * *jconf*, char * *name*)

jconf 内の SESARCH モジュール設定構造体を名前で検索する.

引数:

jconf [in] global configuration

name [in] SEARCH module name

戻り値:

the found SEARCH configuration, or NULL if not found.

jfunc.c の 939 行で定義されています。

6.8.2.22 JCONF_SEARCH* j_get_searchconf_by_id (Jconf * *jconf*, int *id*)

jconf 内の SEARCH モジュール設定構造体を ID で検索する.

引数:

jconf [in] global configuration
id [in] SEARCH module ID

戻り値:

the found SEARCH configuration, or NULL if not found.

jfunc.c の 973 行で定義されています。

6.8.2.23 boolean j_jconf_finalize (Jconf * *jconf*)

jconf 設定パラメータを最終的に決定する

この関数は，jconf ファイルやコマンドオプションによって与えられた jconf 内のパラメータについて精査を行う. 具体的には，値の範囲のチェッ クや，競合のチェック，設定から算出される各種パラメータの計算，使用 するモデルに対する指定の有効性などをチェックする.

この関数は，アプリケーションによって jconf の各値の指定が終了した直後， エンジンインスタンスの作成やモデルのロードが行われる前に呼び出される べきである.

引数:

jconf [i/o] global jconf configuration structure

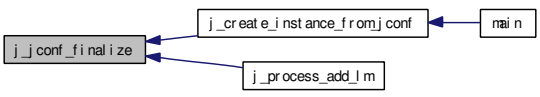
戻り値:

TRUE when all check has been passed, or FALSE if not passed.

m_chkparam.c の 95 行で定義されています。

参照元 j_create_instance_from_jconf(), と j_process_add_lm().

呼出しグラフ:



6.9 LM/AM/SR instance API

LM/AM/SR instance API のコラボレーション図

モデルモジュールやプロセスを直接扱う関数.

関数

- Recog * j_recog_new ()
  エンジンインスタンスを新たにメモリ割り付けする.
- void j_recog_free (Recog *recog)
  エンジンインスタンスを開放する
- Recog * j_create_instance_from_jconf (Jconf *jconf)
  与えられた設定インスタンス内の情報に従って，新たな エンジンインスタンスを 起動・生成する.
- boolean j_load_am (Recog *recog, JCONF_AM *amconf)
  音響モデルを読み込む．
- boolean j_load_lm (Recog *recog, JCONF_LM *lmconf)
  言語モデルを読み込む．
- boolean j_load_all (Recog *recog, Jconf *jconf)
  全てのモデルを読み込み，認識の準備を行なう.
- boolean j_launch_recognition_instance (Recog *recog, JCONF_SEARCH *sconf)
  認識処理インスタンスを立ち上げる．
- boolean j_final_fusion (Recog *recog)
  全てのロードされたモデルと設定からエンジンインスタンスを 最終構成する.

6.9.1 説明

モデルモジュールやプロセスを直接扱う関数.

6.9.2 関数

6.9.2.1 Recog* j_recog_new ()

エンジンインスタンスを新たにメモリ割り付けする.

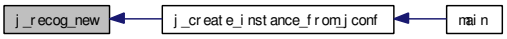
戻り値:

the newly allocated engine instance.

instance.c の 746 行で定義されています。

参照元 j_create_instance_from_jconf().

呼出しグラフ:



6.9.2.2 void j_recog_free (Recog * *recog*)

エンジンインスタンスを開放する

インスタンス内でこれまでにアロケートされた全てのメモリも開放される.

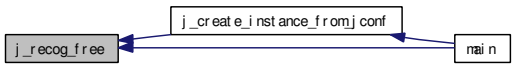
引数:

recog [in] engine instance.

instance.c の 800 行で定義されています。

参照元 j_create_instance_from_jconf(), と main().

呼出しグラフ:



6.9.2.3 Recog* j_create_instance_from_jconf (Jconf * *jconf*)

与えられた設定インスタンス内の情報に従って，新たな エンジンインスタンスを 起動・生成する.

設定インスタンス内のパラメータのチェック後，モデルを読み込み，木構 造化辞書の生成，ワークエリアおよびキャッシュの確保などを行う．A/D-in の初期化以外で認識を開始するのに必要な処理をすべて行う.

引数:

jconf [in] gloabl configuration instance

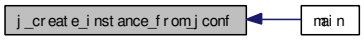
戻り値:

the newly created engine instance.

jfunc.c の 661 行で定義されています。

参照元 main().

呼出しグラフ:



6.9.2.4 boolean j_load_am (Recog * *recog*, JCONF_AM * *amconf*)

音響モデルを読み込む.

この関数は，与えられた AM 設定に従って AM 処理インスタンスを生成し， その中に音響モデルをロードします．その後，その AM 処理インスタンスは 新たにエンジンインスタンスに登録されます．AM 設定はこの関数を 呼ぶ前にあらかじめ全体設定 recog->jconf に登録されている必要があります.

引数:

recog [i/o] engine instance
amconf [in] AM configuration to load

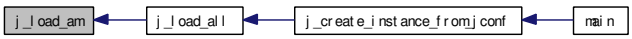
戻り値:

TRUE on success, or FALSE on error.

m_fusion.c の 428 行で定義されています。

参照元 j_load_all().

呼出しグラフ:



6.9.2.5 boolean j_load_lm (Recog * *recog*, JCONF_LM * *lmconf*)

言語モデルを読み込む.

この関数は，与えられた LM 設定に従って LM 処理インスタンスを生成し， その中に言語モデルをロードします．その後，その LM 処理インスタンスは 新たにエンジンインスタンスに登録されます．LM 設定はこの関数を 呼ぶ前にあらかじめ全体設定 recog->jconf に登録されている必要があります.

辞書の読み込み時にトライフォンへの変換および音響モデルとのリンクが 同時に行われます．このため，この言語モデルが使用する音響モデルの インスタンスを引数 am として指定する必要があります.

引数:

recog [i/o] engine instance
lmconf [in] LM configuration to load

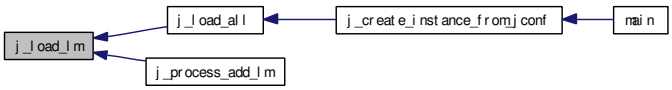
戻り値:

TRUE on success, or FALSE on error.

m_fusion.c の 517 行で定義されています。

参照元 j_load_all(), と j_process_add_lm().

呼出しグラフ:



6.9.2.6 boolean j_load_all (Recog * *recog*, Jconf * *jconf*)

全てのモデルを読み込み，認識の準備を行なう.

この関数では，jconf 内にある（複数の） AM 設定パラメータ構造体や LM 設定パラメータ構造体のそれぞれに対して，AM/LM 処理インスタンスを生成 する．そしてそれぞれのインスタンスについてその中にモデルを読み込み， 認識用にセットアップする．GMM もここで読み込まれる.

引数:

recog [i/o] engine instance
jconf [in] global configuration variables

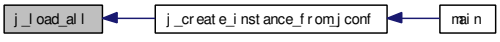
戻り値:

TRUE on success, FALSE on failure.

m_fusion.c の 615 行で定義されています。

参照元 j_create_instance_from_jconf().

呼出しグラフ:



6.9.2.7 boolean j_launch_recognition_instance (Recog * *recog*, JCONF_SEARCH * *sconf*)

認識処理インスタンスを立ち上げる.

この関数は，与えられた SEARCH 設定に従って 認識処理インスタンスを生成し， 対応する音声認識器を構築します．その後，その生成された認識処理インスタンスは 新たにエンジンインスタンスに登録されます．SEARCH 設定はこの関数を 呼ぶ前にあらかじめ全体設定 jconf に登録されている必要があります．

引数:

recog [i/o] engine instance
sconf [in] SEARCH configuration to launch

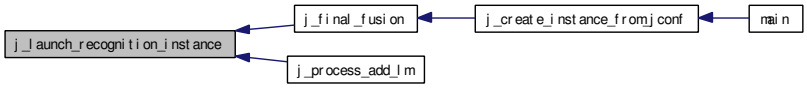
戻り値:

TRUE on success, or FALSE on error.

m_fusion.c の 865 行で定義されています。

参照元 j_final_fusion(), と j_process_add_lm().

呼出しグラフ:



6.9.2.8 boolean j_final_fusion (Recog * *recog*)

全てのロードされたモデルと設定からエンジンインスタンスを 最終構成する．

この関数は，認識準備のための最終処理を行う．内部では，

- 必要な MFCC 計算インスタンスの生成
- 指定された LM/AM の組からの認識処理インスタンス生成
- モデルに依存する認識用パラメータの設定
- 第 1 パス用の木構造化辞書を認識処理インスタンスごとに構築
- 認識処理用ワークエリアとキャッシュエリアを確保
- フロントエンド処理のためのいくつかの値とワークエリアの確保

を行う．この関数が終了後，エンジンインスタンス内の全てのセットアップ は終了し，認識が開始できる状態となる．

この関数は，j_jconf_finalize() と j_load_all() が終わった状態で 呼び出す必要がある．呼出し前には，recog->jconf に (j_load_all でともに 使用した) jconf を格納しておくこと．

引数:

recog [in] engine instance

戻り値:

TRUE when all initialization successfully done, or FALSE if any error has been occured.

m_fusion.c の 1111 行で定義されています。

参照元 j_create_instance_from_jconf().

呼出しグラフ: